Implementation of curved shape grammars

Implementation of curved shape grammars

Iestyn Jowers

School of Mechanical Engineering, University of Leeds, Leeds, LS2 9JT, UK

e-mail: i.jowers@leeds.ac.uk

Christopher Earl

Faculty of Mathematics, Computing and Technology, The Open University, Milton Keynes, MK7 6AA, UK

e-mail: c.f.earl@open.ac.uk

Implementation of curved shape grammars

Abstract

Research into shape grammar implementation has largely been concerned with rectilinear shapes and there has been little research concerning implementation on curved shapes. This reflects developments of the shape grammar formalism which has traditionally been defined according to straight lines. In this paper, issues regarding the application of shape grammars on curved shapes are investigated. This investigation builds on algorithms for implementing shape operations on curved shapes, in which the embedding properties of parametric curves are compared according to their intrinsic properties. In the paper, the algorithms are implemented in a shape grammar interpreter that enables the application of shape grammars on shapes composed of quadratic Bézier curves. The interpreter is illustrated via application of a shape grammar that generates Celtic knotwork patterns. Implementing shape grammars on curved shapes highlights difficulties that arise when the shape grammar formalism is applied to curved shapes, and the paper concludes with a discussion that explores these difficulties.

1. Introduction

Traditionally, shape grammars have been concerned with rectilinear shapes, such as those composed of straight lines or planes. As a result, issues inherent in extending the shape grammar formalism to include curved shapes have received little attention. Despite this, applications of shape grammars on curved shapes have been introduced, for example Orsborn et al. (2006) explore the application of shape grammars to the generation of cross-over vehicle designs. Similarly, implementations have been introduced that enable the application of shape grammars on curved shapes, for example McCormack and Cagan (2003) describe a shape grammar interpreter in which the application of shape operations is facilitated through the definition of representative straight-line shapes. While this research illustrates the potential for the shape grammar formalism to extend beyond its traditional rectilinear roots, the implication of applying shape grammars to curved shapes and the underlying issues that arise when considering the implementation of such shape grammars remain to be explored.

In a recent paper, Jowers and Earl (2010) presented an approach for addressing the problem of shape grammar implementation on curved shapes that is based on the intrinsic comparison of parametric curves. This method of comparison ensures that shapes operations are applied to curved shapes based on the embedding properties of their constituent curves, defined under Euclidean transformations. Jowers an Earl introduced shape algorithms that enable implementation of shape operations on shapes composed of parametric curve segments, and in this paper these algorithms are investigated with the aim to define a shape grammar interpreter for shapes composed of quadratic Bézier curves arranged in a plane. Consideration of a specific representation of curves makes it possible to define and implement a shape grammar interpreter, and quadratic Bézier curves are particular suitable since they are the simplest of a class of curves that is in common use in computer aided geometric design (Farin et al., 2002). The resulting interpreter is illustrated via application of a shape grammar that generates Celtic knotwork patterns.

The paper concludes with a discussion that explores the implications of applying shape grammars to curved shapes, and of the issues that arise when considering the implementation of curved shape grammars. In particular, the differences in embedding properties between rectilinear shapes and curved shapes are highlighted. These differences raise difficulties with respect to implementing shapes grammars on curved shapes and merit further investigation.

2. Background

2.1 Shape Grammars

A shape grammar (Stiny, 2006) consists of an initial shape and a set of rules of the form $\alpha \rightarrow \beta$, where α and β are both shapes, as illustrated in figure 1. In this context, a shape is defined visually, as a finite arrangement of *geometric elements*, such as lines or curves, each with a definite boundary and limited but non-zero extent. A rule is applicable to a shape γ if some similarity transformation (typically a Euclidean transformation) of the shape α on the left hand side of the rule is a subshape of γ. Application of the rule removes the transformed instance of the subshape α and replaces it with a similarly transformed instance of the shape β on the right hand side of the rule. For example, the shape rule in figure 1 removes a lens and replaces it with a translated lens, as indicated by the coordinate axis. Repeated application of shape rules to an initial shape leads to the generation of a sequence of designs, as illustrated in figure 2.

Figure 1 – an example shape rule

Shape grammars embody the philosophy that a designer ought to be able to recognise and manipulate any subshape or structure that can be perceived within a shape. As a result, application of a shape rule is not restricted to the geometric elements initially used to define a design; instead they are applicable to any subshapes that can be perceived to be embedded in the design. Indeed, within the shape grammar formalism the structure of a design is defined retrospectively according to shape rule applications (Stiny, 2006). A consequence of this is that shape grammars often generate designs that incorporate some unexpected results that follow from the recognition and manipulation of new interpretations of shapes. For example, the design sequence in figure 2, begins with an initial shape, and ends with a shape that is a rotation of this initial shape, but this sequence is generated via application of a rule that merely translates a subshape of the initial shape (figure 1). This unexpected result occurs because after a single application of the shape rule additional instances of the lens in the left hand side of the rule have emerged. Further applications of the shape rule can now be used to recognise and manipulate these emergent shapes, and restructure the design so that it appears to be rotated.

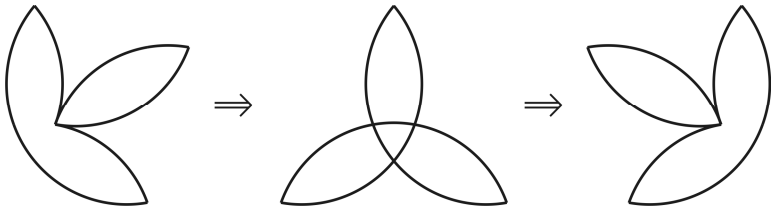


Figure 2 – example design sequence

Shape grammars have been found to be a powerful generative mechanism and have been employed in a range of design disciplines, from architecture, e.g. Koning and Eizenberg (1981), to engineering, e.g. Brown and Cagan (1997). However, few of these applications are computationally implemented and instead it is common for rules to be applied and designs generated as a paper based exercise. This is because the task of developing a shape grammar interpreter – a computational system intended to automate the application of shape rules – is not trivial, requiring a solution to the subshape detection problem. For shapes composed of straight lines a solution to this problem was presented by Krishnamurti (1981), based on a method of maximal lines. This approach has been implemented in a variety of shape grammar interpreters, such as those by Chase (1989), Tapia (1999) and Trescak et al.

(2009). However, an obvious limitation of the approach is that it can only generate designs composed of rectilinear shapes.

An interpreter for shapes composed of freeform curves was introduced by McCormack and Cagan (2003). In the interpreter, curved shapes are associated with representative shapes composed of straight lines and subshape detection is facilitated via comparison of these rectilinear shapes. Consequently, although the designs generated by the interpreter are composed of curve segments, subshape detection is concerned with the embedding properties of the rectilinear shapes, resulting in the possibility that the embedding properties of the original curved designs are neglected. An alternative method of implementing curved subshape detection was presented by Jowers and Earl (2010). This method is based on an intrinsic comparison of shapes composed of parametric curves in which the embedding properties of shapes are compared directly, rather than according to representative rectilinear shapes. This method has the potential to increase the scope of shape grammars applications to design disciplines where rectilinear shapes are not sufficient, such as product design, and its implementation is investigated in this paper.

2.2 Intrinsic Comparison of Curved Shapes

Intrinsic comparison of parametric curves provides a method for determining whether or not two curved shapes can be mapped onto each other under a Euclidean transformation. It forms the basis of shape algorithms introduced by Jowers and Earl (2010) that define the operations necessary to implement a shape grammar interpreter for shapes composed of parametric curves in two- or three-dimensional space. In particular, the algorithms define operations that enable the recognition of subshapes embedded in a curved shape under Euclidean transformations, along with operations that allow the recognised subshapes to be removed and replaced according to shape rules.

Intrinsic comparison is based on the fundamental theorem of space curves which states that any two continuous functions of a real variable define a space curve, and serve as its curvature and torsion, with the real variable as a natural parameter of the curve (Do Carmo, 1976). Taking this theorem as a starting point, Jowers and Earl (2010) showed that if two parametric curves have curvature and torsion functions that can be mapped onto each other under a scaling factor and a reparameterisation function, then the curves can be mapped onto each under a Euclidean transformation. This is formally expressed as follows: given two space curves C_1 and C_2, parameterised according to

arbitrary parameters t and u respectively, the curves can be mapped onto each other under a Euclidean transformation if and only if

$$\kappa_1(t) = \lambda^{-1}\kappa_2(u(t)) \qquad \text{— (1)}$$

$$\tau_1(t) = \lambda^{-1}\tau_2(u(t)) \qquad \text{— (2)}$$

where κ_i and τ_i are the curvature and torsion of the curve C_i, respectively, λ ($\neq 0$) is a scaling factor and $u = u(t)$ is a continuous reparameterisation function. Here, the parametric curves under consideration are of infinite extent and act as *descriptors* of the curve segments that are used to construct shapes.

Geometrically, comparison of the intrinsic properties of curve segments provides a means of comparing the shape of their descriptors, and shape operations are applicable only between curve segments that have descriptors of the same shape. Shape operations also consider the relative position of curve segments with respect to their descriptors, and this can be determined by comparing their end points under the reparameterisation function $u = u(t)$. In particular, if two curve segments $\mathbf{x}_1(t)$ and $\mathbf{x}_2(u)$ have descriptors of the same shape then equations (1) and (2) hold for some constant λ ($\neq 0$) and some function $u = u(t)$. Then, consideration of the end points of $\mathbf{x}_1(t)$ and $\mathbf{x}_2(u)$ according to the function $u = u(t)$, will determine their relative positions, as illustrated in figure 3. Here, there are three distinct cases to consider: firstly, the end points can overlap or are coincident so that $\mathbf{x}_1(t)$ and $\mathbf{x}_2(u)$ have one or more points in common, as illustrated in figure 3a – d; secondly, the end points of one curve segment can lie within the end points of the second so that one segment is completely embedded in the second, as illustrated in figure 3e – h; finally, for any other arrangement of end points, $\mathbf{x}_1(t)$ and $\mathbf{x}_2(u)$ are completely disjoint.

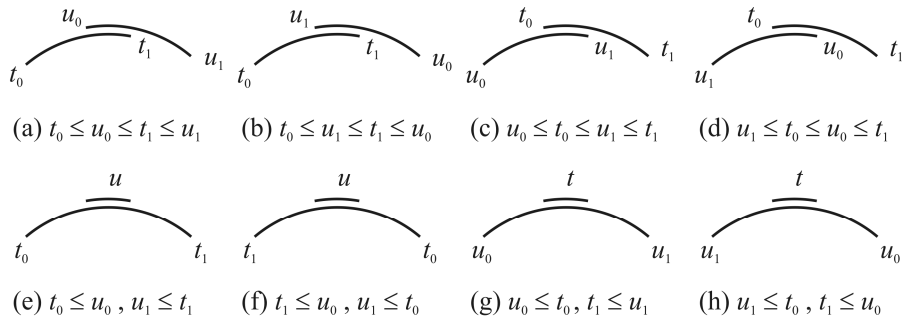


Figure 3 – overlapping curve segments

The shape algorithms introduced by Jowers and Earl (2010) use intrinsic comparison of parametric curves in the implementation of shape operations. These enable the application of shape rules on curved shapes. For example, a rule of the form $\alpha \rightarrow \beta$ is applied to the shape γ by i) reducing γ

to its maximal representation; ii) detecting subshapes $t(\alpha)$ of γ that are Euclidean transformations of the shape α; iii) removing an instance of $t(\alpha)$; and iv) adding the shape $t(\beta)$ which is the shape β under the same Euclidean transformation.

Under a maximal representation, shapes are composed of the smallest possible set of curve segments. This ensures a canonical representation and simplifies the application of shape operations. The maximal representation of a shape γ results from merging any curve segments that have the same descriptor and that have any points in common. In the shape algorithms it is calculated by comparing pairs of curve segments according to equations (1) and (2) to determine if they have the same descriptor, and then comparing their end points to determine if they can be merged to form a single curve, as illustrated in figure 3a – d, or if one curve can be removed without causing any visual alterations to the shape, as illustrated in figure 3e – h. Similarly, subshape detection is implemented by comparing the curves in the shape α to the maximal curves in the shape γ according to equations (1) and (2) and according to their end points. If a shape α is a subshape of a shape γ then there exists a Euclidean transformation t such that all of the curves that compose the shape $t(\alpha)$ are embedded in the maximal curves that compose the shape γ, as illustrated in figure 3e – h. Application of a rule involves removing all the curves that compose the shape $t(\alpha)$ from the maximal curves that compose the shape γ, resulting in the shape $\gamma - t(\alpha)$. Removal of one shape from another involves application of the shape difference operator on the curve segments that composed the shapes. Application of the rule is completed by adding the shape $t(\beta)$, resulting in the shape $[\gamma - t(\alpha)] + t(\beta)$. Addition of shapes involves application of the shape union operation which is simply applied by adding the curve segments that compose one shape to the set of curve segments that compose the other.

As discussed, implementation of the shape algorithms introduced by Jowers and Earl (2010) is dependent on the intrinsic comparison of parametric curves expressed in equations (1) and (2). This comparison is defined according to the non-zero constant λ and the reparameterisation function $u = u(t)$. However, in Jowers and Earl (2010), a general method for determining these, or of proving their existence, was not investigated. Indeed, it was suggested that λ and $u = u(t)$ are dependent on the particular parametric curves used to compose a shape. In the next two sections the application of these algorithms is explored for shapes composed of quadratic Bézier curves. Consideration of a specific representation of curves makes it possible to determine explicit expressions for λ and $u = u(t)$, and therefore implement the algorithms in a shape grammar interpreter. In the next section, further details

regarding implementing the algorithms are explored and a shape grammar interpreter for curved shapes is introduced. Then, in Section 4, the interpreter is illustrated via application of a shape grammar that generates Celtic knotwork patterns.

3. Implementing Curved Shape Grammars

3.1 Intrinsic Comparison of Quadratic Bézier Curves

A quadratic Bézier curve is a parametric curve of order two that is defined according to three control points, $\mathbf{b}_0$, $\mathbf{b}_1$ and $\mathbf{b}_2$, as illustrated in figure 4.

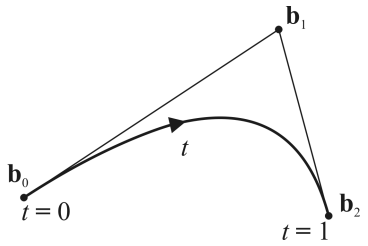


Figure 4 – a quadratic Bézier curve

It is a parabolic curve defined according to a parameter t over the interval $0 \leq t \leq 1$ and can be formally expressed by the polynomial

$$\mathbf{B}(t) = \mathbf{a}t^2 + \mathbf{b}t + \mathbf{c} \qquad \text{— (3)}$$

where $\mathbf{a} = (a_x, a_y)$, $\mathbf{b} = (b_x, b_y)$ and $\mathbf{c} = (c_x, c_y)$ are defined by the control points of the curve.

As discussed in Section 2.2, intrinsic comparison of parametric curves involves a comparison of their curvature and torsion functions according to equations (1) and (2). For quadratic Bézier curves intrinsic comparison is achieved by comparing only the curvature function, according to equation (1), this is because they are planar curves and their torsion is, by definition, zero. Consider two quadratic Bézier curves $\mathbf{B}_1(t)$ and $\mathbf{B}_2(u)$, defined according to arbitrary parameters t and u respectively, where $0 \leq t \leq 1$ and $0 \leq u \leq 1$. From equation (1), the descriptors of $\mathbf{B}_1(t)$ and $\mathbf{B}_2(u)$ can be mapped onto each other under a Euclidean transformation if and only if

$$\kappa_1(t) = \lambda^{-1}\kappa_2(u(t))$$

for some constant λ ($\neq 0$) and some continuous reparameterisation function $u = u(t)$. $\mathbf{B}_1(t)$ and $\mathbf{B}_2(u)$ are both defined by quadratic polynomial vectors, it therefore follows that the reparameterisation function, $u = u(t)$, must be a linear function of the form

$$u(t) = \mu t + \nu$$

for some constants μ ($\neq 0$) and v. Consequently, the descriptors of $\mathbf{B}_1(t)$ and $\mathbf{B}_2(u)$ can be mapped onto each other under a Euclidean transformation if and only if

$$\kappa_1(t) = \lambda^{-1}\kappa_2(\mu t + v) \qquad \text{— (4)}$$

for some constants λ ($\neq 0$), μ ($\neq 0$) and v. Expressions for λ, μ and v can be derived by introducing the explicit functions for $\kappa_1(t)$ and $\kappa_2(\mu t + v)$. For a two-dimensional parametric curve $\mathbf{C}(t) = (x(t), y(t))$, curvature is given by

$$\kappa = \frac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{\left(\dot{x}^2 + \dot{y}^2\right)^{3/2}}$$

where the dot implies differentiation with respect to t. From this, the curvature function of a quadratic Bézier curve $\mathbf{B}(t)$ is given by

$$\kappa(t) = \frac{D}{\left(At^2 + Bt + C\right)^{3/2}} \qquad \text{— (5)}$$

where

$$A = 4a_x^2 + 4a_y^2 \qquad \text{— (6a)}$$

$$B = 4a_x b_x + 4a_y b_y \qquad \text{— (6b)}$$

$$C = b_x^2 + b_y^2 \qquad \text{— (6c)}$$

$$D = 2a_y b_x - 2a_x b_y \qquad \text{— (6d)}$$

Introducing explicit functions for $\kappa_1(t)$ and $\kappa_2(\mu t + v)$ into equation (4) gives

$$\frac{D_1}{\left[A_1 t^2 + B_1 t + C_1\right]^{3/2}} = \frac{D_2}{\lambda\left[A_2(\mu t + v)^2 + B_2(\mu t + v) + C_2\right]^{3/2}}$$

where A_i, B_i, C_i and D_i (i = 1, 2) refer to the values A, B, C, and D defined for a curve $\mathbf{B}_i$ according to equations (6a) - (6d). Rearranging this expression gives

$$(\lambda D_1)^{2/3}[A_2\mu^2 t^2 + (2A_2\mu v + B_2\mu)t + A_2 v^2 + B_2 v + C_2] = (D_2)^{2/3}[A_1 t^2 + B_1 t + C_1]$$

which is true for all values of t. According the coefficients for t^2, t^1 and t^0 can be equated, resulting in a series of three equations which can be solved for λ, μ and v to give

$$\lambda = \pm\frac{A_2^{3/2} D_2}{A_1^{3/2} D_1}\left(\frac{B_1^2 - 4A_1C_1}{B_2^2 - 4A_2C_2}\right)^{3/2}$$

$$\mu = \pm\frac{A_1}{A_2}\left(\frac{B_2^2 - 4A_2C_2}{B_1^2 - 4A_1C_1}\right)^{1/2}$$

$$\nu = -\frac{1}{2A_2}\left(B_2 \mp B_1\left(\frac{B_2^2 - 4A_2C_2}{B_1^2 - 4A_1C_1}\right)^{1/2}\right)$$

Further, from equations (5a) - (5d) it can be shown that

$$B^2 - 4AC = -4D^2$$

and as a result λ, μ and ν are simply defined as follows

$$\lambda = \pm\frac{A_2^{3/2}D_1^2}{A_1^{3/2}D_2^2} \qquad \text{— (7)}$$

$$\mu = \pm\frac{A_1D_2}{A_2D_1} \qquad \text{— (8)}$$

$$\nu = \frac{\pm B_1D_2 - B_2D_1}{2A_2D_1} \qquad \text{— (9)}$$

Equations (7) – (9) provide explicit definitions for λ, μ and ν that enable intrinsic comparison of quadratic Bézier curves according to equation (4). Using these equations it is possible to implement the shape algorithms introduced by Jowers and Earl (2010) on shapes composed of quadratic Bézier curves. However, it is possible that equations (7) – (9) are unsolvable for λ, μ and ν. In particular if A_i = 0 or D_i = 0 (i = 1, 2), then there will be no solution. These scenarios can be explained geometrically by examining each case in turn. If D = 0 for a curve then, from equation (5), its curvature is zero and the curve is by definition a straight line. Similarly, if A = 0 for a curve then, from equation (6a) $4a_x^2 + 4a_y^2 = 0$, which has the real solution a_x = 0 and a_y = 0, i.e. **a** = 0. As a result, from equation (3), a curve $\mathbf{B}(t)$ for which A = 0 is given by $\mathbf{B}(t) = bt + c$, which is the equation of a straight line.

From this analysis we have two distinct cases. In the first case, Equations (7) – (9) provide explicit definitions for λ, μ and ν. This is because, in this case, $\mathbf{B}_1(t)$ and $\mathbf{B}_2(u)$ are both non-degenerate quadratic Bézier curves which are, by definition, parabolic curve segments. As a result, the descriptors of the curves are parabolas which can be mapped onto each other under a Euclidean transformation. In the second case, Equations (7) – (9) cannot be solved for λ, μ and ν. This is because, in this case, either $\mathbf{B}_1(t)$ or $\mathbf{B}_2(u)$, or both, are degenerate quadratic Bézier curves defined by a straight line segment. As discussed in Jowers and Earl (2010), intrinsic comparison is not applicable to straight lines. This is because intrinsic comparison of parametric curves uses the varying nature of the intrinsic properties of the curve in order to determine their embedding properties. For geometric elements that have constant intrinsic properties, such as straight lines, intrinsic comparison fails because there is no variety with

which to compare. Instead, for such geometric elements it is necessary to use alternative shape algorithms, such as those presented by Krishnamurti (1981).

3.2 Shape Operations Applied to Bézier Curves

The Bézier representation supports some powerful techniques with respect to manipulation of parametric curves and these can be applied in the implementation of shape operations, including sum, difference and subshape detection. Of particular note is the de Casteljau algorithm, which enables the evaluation of the coordinates a point on a Bézier curve, specified by a parameter value. Typically, the de Casteljau algorithm is used to computationally render a Bézier curve, and is used as a means for sub-dividing the curve repeatedly until it can be approximated by straight lines. Here, the technique is used in order to implement shape operations on shapes composed of Bézier curves.

Let $\mathbf{B}(t)$ be a Bézier curve of order n, specified according to the control points $\mathbf{b}_0$, $\mathbf{b}_1$, ..., $\mathbf{b}_n$. A point on the curve is specified according to a parameter value t, where $0 \leq t \leq 1$, and the de Casteljau algorithm evaluates this point according to the recursive formula

$$\begin{cases} \mathbf{b}_i^0 = \mathbf{b}_i \\ \mathbf{b}_i^j = (1-t)\mathbf{b}_i^{j-1} + t\mathbf{b}_{i+1}^{j-1} \end{cases}$$

where $j = 1, 2, ..., n$ and $i = 0, 1, ..., n - j$. This formula produces a triangular set of points, culminating in the point $\mathbf{b}_0^n$, which is the point on $\mathbf{B}(t)$ specified by the parameter value t. For example, for a quadratic Bézier curve $n = 2$ and the recursive formula defines the following triangular set of points

$$\begin{array}{lll} \mathbf{b}_0^0 & \mathbf{b}_1^0 & \mathbf{b}_2^0 \\ \mathbf{b}_0^1 & \mathbf{b}_1^1 & \\ \mathbf{b}_0^2 & & \end{array}$$

Geometrically, the algorithm evaluates the point $\mathbf{b}_0^n$ via repeated linear interpolation of the control polygon according to the ratio $t : (1 - t)$. For example, the point $\mathbf{b}_0^2$ at $t = 1/3$, can be evaluated on a quadratic curve by recursively applying linear interpolation according to the ratio 1/3 : 2/3, as illustrated in figure 5. As a consequence of this process additional points are evaluated and subsets of these points define two curves of order n that result from sub-dividing the original curve at the point t. For example, in figure 5, evaluation of the point $\mathbf{b}_0^2$ at $t = 1/3$, also resulted in the evaluation of the points $\mathbf{b}_0^1$ and $\mathbf{b}_1^1$. These points define two new quadratic Bézier curves. The points $\mathbf{b}_0$, $\mathbf{b}_0^1$ and $\mathbf{b}_0^2$

define the curve segment that extends from $t = 0$ to $t = 1/3$, and the points $\mathbf{b}_0^2$, $\mathbf{b}_1^1$ and $\mathbf{b}_2$ define the curve segment that extends from $t = 1/3$ to $t = 1$.

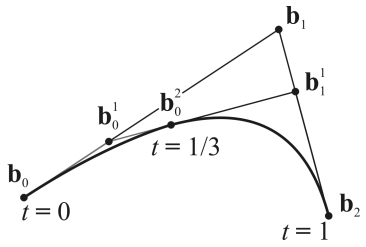


Figure 5 – application of the de Casteljau algorithm at the point $t = 1/3$

If the point to be evaluated sits outside the curve segment, then it is specified according to a parameter value t where $t < 0$ or $t > 1$. In this case, the additional points that are evaluated by the de Casteljau algorithm define two curves that result from extending the original curve to the point t. For example, evaluation of the point specified by the parameter value $t = 4/3$ on the curve in figure 4 is illustrated in figure 6). Now, the points $\mathbf{b}_0$, $\mathbf{b}_1^1$ and $\mathbf{b}_0^2$ define the control points of the extended curve segment, and the points $\mathbf{b}_0^2$, $\mathbf{b}_1^1$ and $\mathbf{b}_2$ define the control points of the curve segment that is added onto the original curve.

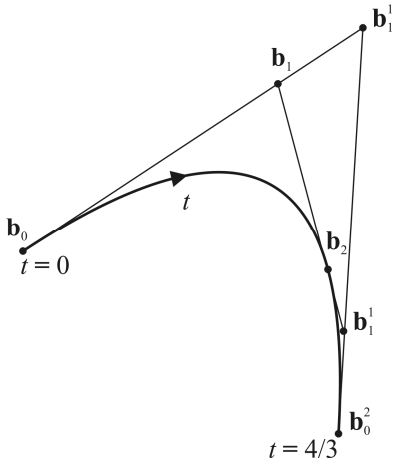


Figure 6 – application of the de Casteljau algorithm at the point $t = 4/3$

With respect to implementing shape operations on shapes composed of Bézier curves, the de Casteljau algorithm provides a procedure for adding and subtracting curve segments, and is particularly useful in the implementation of algorithms for reducing a shape to its maximal representation, and algorithms for calculating shape difference. As discussed in Section 2.2, in order to reduce a curved shape to its maximal representation it is necessary to merge any curved segments that overlap so that they form single curve segments. The de Casteljau algorithm provides a method for reproducing this

result for Bézier curves. As illustrated in figure 6, the de Casteljau algorithm applied at a point specified by a parameter value t, with $t < 0$ or $t > 1$, calculates the control points of an extended curve. This procedure can be used to calculate the curve that results from merging two overlapping Bézier curves, as illustrated in figure 7. Here, two overlapping Bézier curves $\mathbf{B}_1(t)$ and $\mathbf{B}_2(u)$ are illustrated according to two parallel curves. The curves overlap such that $t_0 \leq u_0 \leq t_1 \leq u_1$, where t_0 and t_1 are the parameter values of the end points of $\mathbf{B}_1(t)$, and u_0 and u_1 are the parameter values of the end points of $\mathbf{B}_2(u)$. The curve that results from merging $\mathbf{B}_1$ and $\mathbf{B}_2$ is calculated by applying the de Casteljau algorithm to the curve $\mathbf{B}_2(u)$ at the point specified by the parameter value $u(t_0)$.

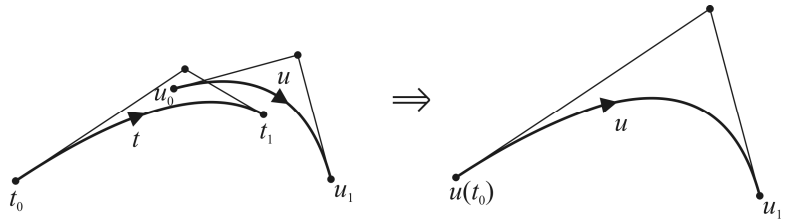


Figure 7 – merging two Bézier curves to form a third curve

Similarly, application of shape difference involves determining the shape that remains after removing an embedded subshape, and for shapes composed of Bézier curves, the de Casteljau algorithm can be used to calculate the curve segments that remain. As illustrated in figure 5, the de Casteljau algorithm applied to a Bézier curve at a point specified by a parameter value t, with $0 < t < 1$, calculates the control points of the curve segments that result from sub-division of the curve at that point. This procedure can be used to reproduce the result of removing embedded curve segments by calculating the segments that will remain and removing the original, as illustrated in figure 8. Here, a Bézier curve $\mathbf{B}_1(t)$ is embedded in a second curve $\mathbf{B}_2(u)$ such that $u_0 \leq t_0 \leq t_1 \leq u_1$, and the curves that result from removing $\mathbf{B}_1(t)$ from $\mathbf{B}_2(u)$ are calculated by applying the de Casteljau algorithm to the curve $\mathbf{B}_2(u)$ at the points specified by the parameter values $u(t_0)$ and $u(t_1)$.

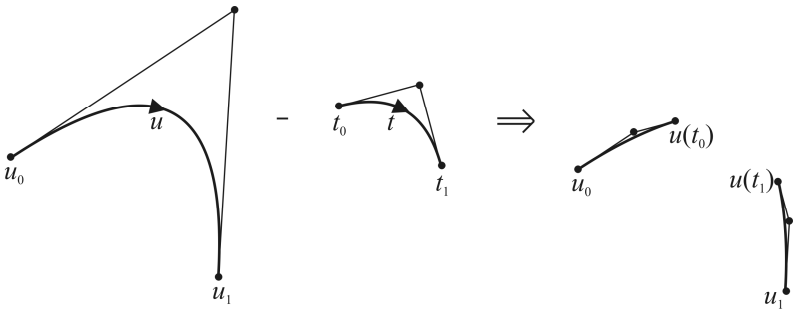


Figure 8 – shape difference applied to Bézier curves

3.3 A Curved Shape Grammar Interpreter

QI (Quad Interpreter), illustrated in figure 9, is a shape grammar interpreter for shapes composed of quadratic Bézier curves arranged in a plane. It is built on the algorithms introduced by Jowers and Earl (2010), which are implemented according to equations (7) – (9), and according to the shape operations described in the previous section. According to the classifications defined by Chase (2002), QI offers a semi-automatic mode of interaction. This means the system calculates possible applications of shape rules and the user selects from these which rule to apply, and where to apply it.

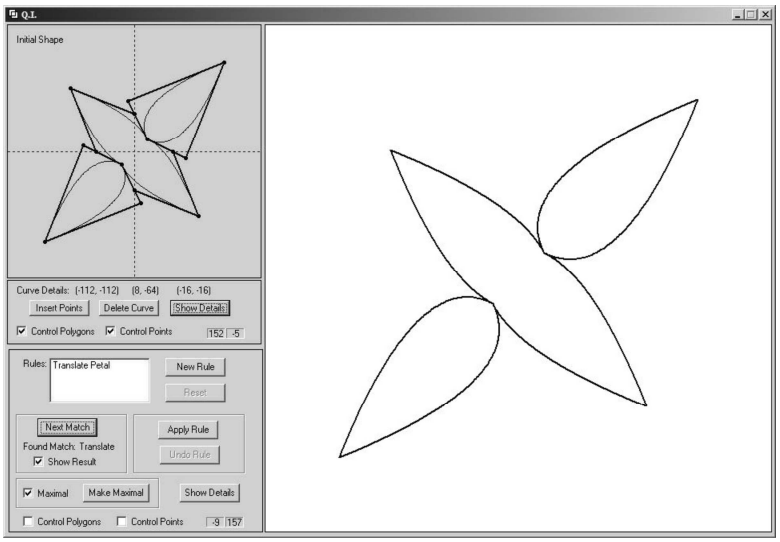


Figure 9 – QI, a shape grammar interpreter for curved shapes

The user interface of QI, illustrated in figure 9, is a dialog box based graphics interface in which shapes and shape rules can be intuitively defined via mouse input. It is segregated into three regions: the biggest of the regions, on the right hand side, is the design window which displays the current design in a generative sequence; the region in the top left corner is a drawing space where the initial shape is defined; the region in the bottom left is a console that provides control over rule application and over the display of shapes and shape rules.

At the start of a generative sequence the shape in the design window is identical to the initial shape. However, this shape cannot be manipulated directly and instead alternative designs are generated via application of shape rules in a shape grammar. The shape rules are of the form $\alpha \rightarrow \beta$ and are defined in a second dialog box interface, as illustrated in figure 10. In this interface, the shape α is defined in the drawing space on the left hand side and the shape β is defined in the drawing space on the right hand side. All shapes in QI are composed of quadratic Bézier curves which are defined via their control polygons, as illustrated in the initial shape window in figure 9. The curve segments are created by

simply clicking in a drawing space to define control points, which can be dragged to their desired location. In order to assist in the composition of shapes and shape rules the drawing spaces all contain a Cartesian coordinate system. For display purposes it is also possible to draw the curves without their control polygons, as illustrated in figure 10.

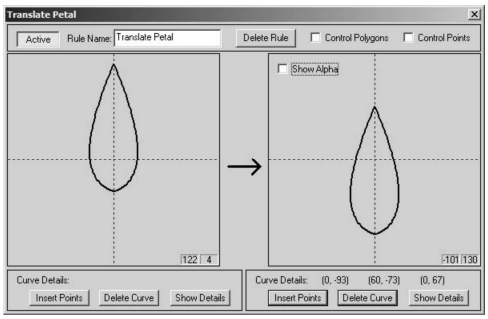


Figure 10 – a shape rule that translates a petal shape

The shape in the design window in figure 9 is defined according to eight curve segments and contains two instances of a petal subshapes that can be recognised and manipulated according to the shape rule in figure 10. In this rule, a petal shape composed of two curve segments is translated along its vertical axis. As discussed in Section 2, a rule $\alpha \rightarrow \beta$ is applied to a shape γ by first finding a similarity transformation of the shape α embedded in γ. This embedded transformation of α is removed from γ and is replaced by the shape β under the same similarity transformation. In QI, these shape operations are applied according to the shape algorithms introduced by Jowers and Earl (2010), which are implemented according to equations (7) – (9), and according to the shape operations described in the previous section. First, the shapes α and γ are reduced to their maximal representations. Then the subshape detection algorithms are applied in order to calculate the possible transformations that embed the shape α in γ. These different transformations can then be scrolled through in order to find the required subshape of γ. For example, application of the subshape detection algorithms on the design in figure 9, with respect to the rule in figure 10, results in the detection of the two petal subshapes illustrated in figure 11. Here, repeated matches that result from reflective symmetry are ignored.

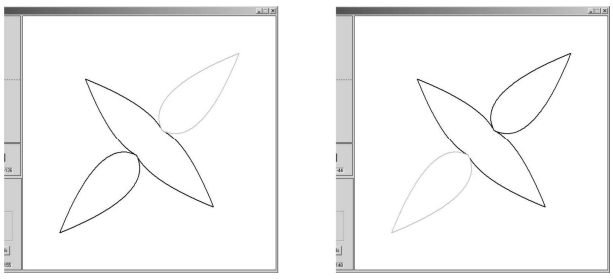


Figure 11 – subshape detection

Once a subshape has been detected it can be replaced according to the shape rule in order to generate new designs in a design sequence. For example, application of the rule in figure 10 to the subshapes illustrated in figure 11 results in the design sequence in figure 12.

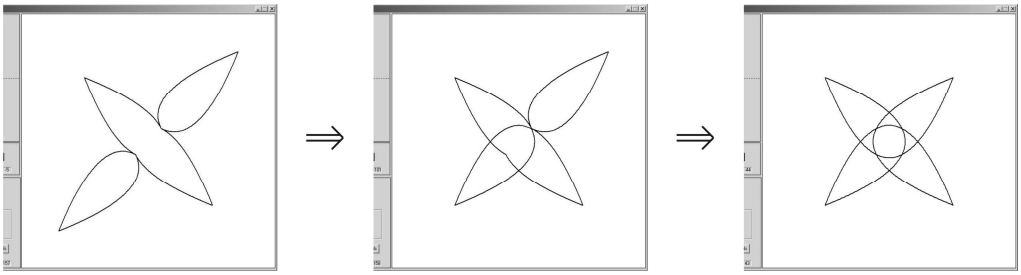


Figure 12 – example design sequence

As the shape rule is applied, new subshapes emerge which can be recognised and manipulated. For example, application of the subshape detection algorithms on the final design in figure 12, with respect to the rule in figure 10, now results in the detection of four petal subshapes, as illustrated in figure 13. Two additional instances of the petal subshape have emerged as a consequence of the previous rule applications. These emergent subshapes can be detected in QI because when the new design is reduced to its maximal representation the eight quadratic Bézier curves that compose the initial shape are merged, resulting in a shape composed of four curve segments.

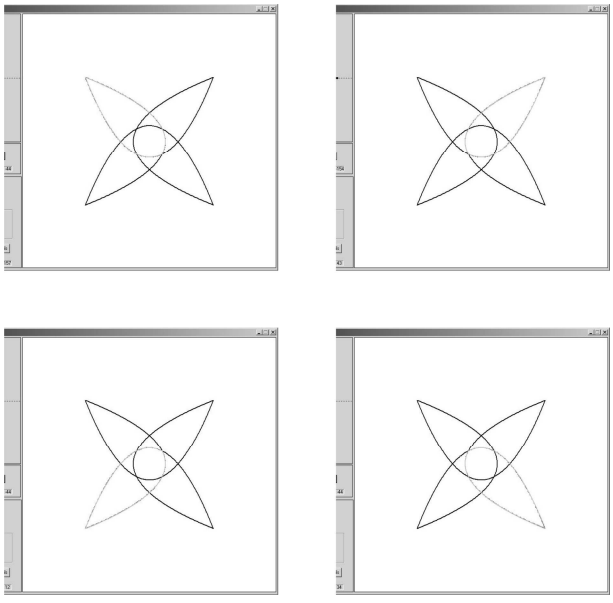


Figure 13 – subshape detection

In addition to the petal shape, other subshapes can be seen to be embedded in this design, and these can also be recognised and manipulated via application of shape rules. For example, in the centre of the shape is a curved ‘square’ which can be recognised and manipulated by defining an appropriate rule,

such as the rule in figure 14. This rule recognises a curved ‘square’ and replaces it with a bigger ‘square’ rotated about its centre by 45º.

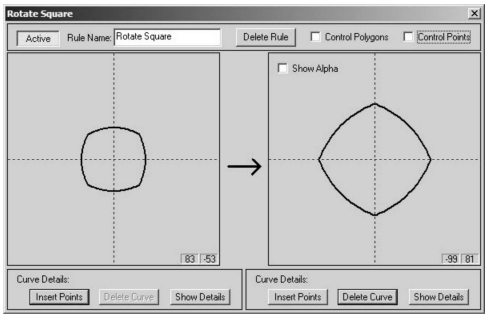


Figure 14 – a shape rule that rotates a curved ‘square’

Application of this curved ‘square’ rule is illustrated in figure 15. Here, the subshape detection algorithms are applied in order to recognise a subshape of the design that is a Euclidean transformation of the ‘square’. Then the ‘square’ is removed and replaced with the rotated ‘square’ according to the shape replacement algorithms.

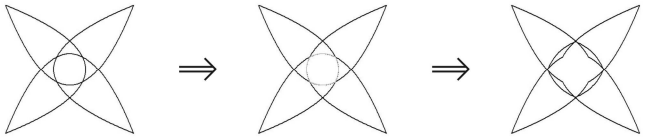


Figure 15 –recognition and replacement of curved ‘square’

4. Implementation of a Celtic Knotwork Grammar

Using QI, a shape grammar was developed that enables the generation of Celtic knotwork designs. These elaborate patterns are found throughout Great Britain and Ireland, and are a predominant motive of the Celtic style of the Christian period (AD 450 – 1066). They were used as ornamentation in a range of media including illuminated manuscripts such as the Lindisfarne Gospels (the British Library, London), sculptured stone monuments such as the Nevern Cross illustrated in figure 16, and metalwork such as the Ardagh Chalice (the National Museum of Ireland, Dublin). The knotwork patterns are an evolution of the regular plaitwork common in Egyptian, Greek and Roman decorative art in which designs represent weaved bands which maintain an over-under alternating pattern (Allen, 1904). Knotwork patterns are derived from this plaitwork via the introduction of *breaks* which enable the development of complicated forms of interlaced ornament.

Figure 16 – Nevern Cross (St Brynach's Church, Nevern, West Wales)

Typically, knotwork patterns are constructed using a grid-based approach, e.g. Allen (1904), Bain (1975), Bain (1990). For example, the construction method described by Bain (1990) is illustrated in figure 17. Here, a grid is divided by a lattice of diagonals which serve as setting-out lines that define the width of the bands composing the knotwork. Features of knotwork patterns such as corners and arches are added by replacing appropriate parts of the lattice, and the remaining lattice is replaced with overlapping plaitwork. This approach can be formally defined according to modular units which are combined strategically to give valid designs, as described by Jablan (2002). Similar approaches have also been used to construct interlacing patterns from other cultures. For example, Knight and Sass (In Press) present a shape grammar for generating Kolam patterns from India that is based on a similar method of combining modular units.

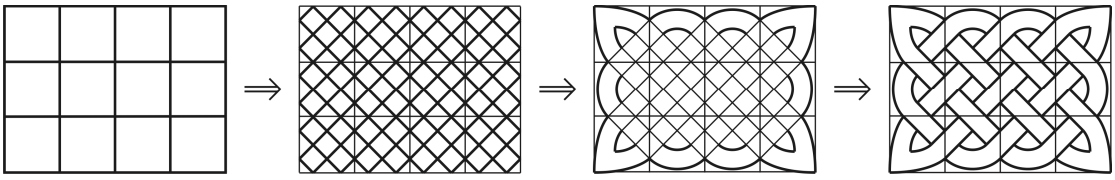


Figure 17 – Bain's method of constructing Celtic knotwork (Bain, 1990)

The Celtic knotwork grammar presented here is based on an alternative method of construction, in which designs are "grown" from a simple initial knot, illustrated in figure 18. The grammar is composed of the 14 rules illustrated in figures 19 and 20, and the inverses of these rules. Here, the inverse of a rule $\alpha \rightarrow \beta$ is defined by taking the shape on the left-hand side and using it on the right-hand side and vice-versa, and is given by $\beta \rightarrow \alpha$. The rules in the grammar are grouped according to whether they apply to the global or local geometry of a design. Global geometry rules serve to define the overall shape of a design. For example, the four global geometry rules illustrated in figure 19 are

used to grow the outer boundary of a design, while the inverses of these rules are used to shrink the outer boundary, or to introduce holes in a design. Local geometry rules are used to define the topology of a design. For example, the ten local geometry rules illustrated in figure 20 are used to change the local topology by introducing breaks in the plaitwork, while the inverses of these rules are used to merge adjacent pieces of band so that they form a plait.

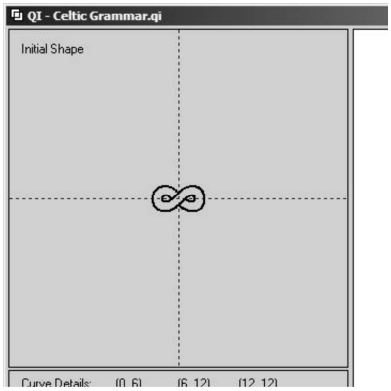
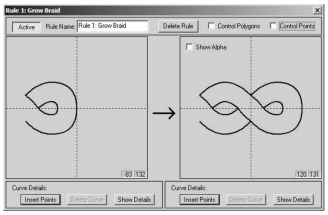
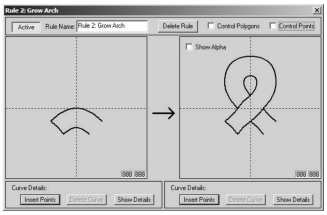


Figure 18 – Initial shape of the Celtic knotwork grammar

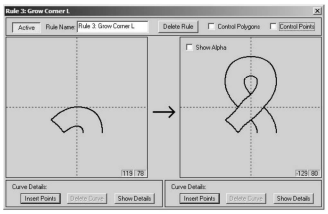
Generation of a Celtic knotwork pattern by "growing" from a initial knot provides significant advantages over the typical grid-based approach illustrated in figure 17 since i) each stage of the generation process is a valid design; and ii) the shape of the final design emerges during the generation process. This is illustrated in figure 21, where an example of the generation process is presented. Here, the constructed knotwork pattern from figure 17 is generated by "growing" from the initial knot. The rules that are applied to move from one design to the next are indicated, and the inverse rules are denoted by the suffix *i*, for example Rule9i.



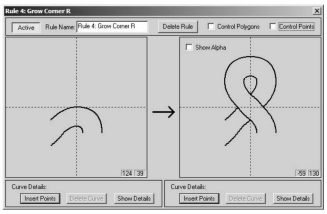
Rule 1: Grow Braid



Rule 2: Grow Arch

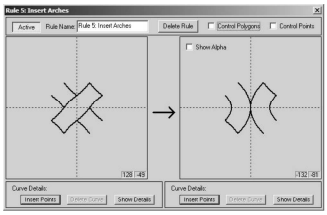


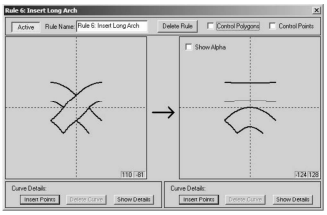
Rule 3: Grow Corner L

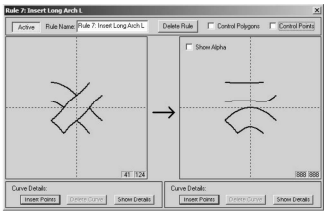


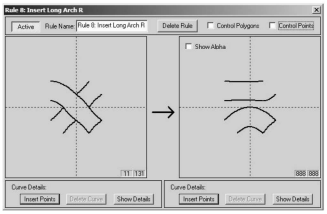
Rule 4: Grow Corner R

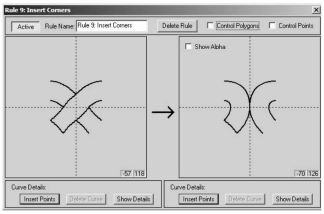
Figure 19 – Global geometry rules of the Celtic knotwork grammar

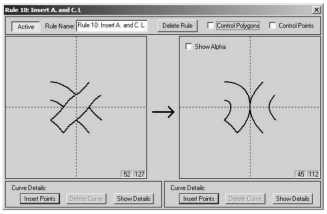

Rule 5: Insert Arches

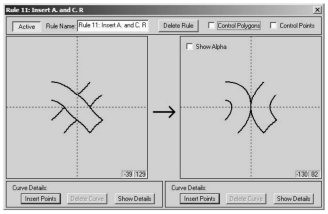

Rule 6: Insert Long Arch

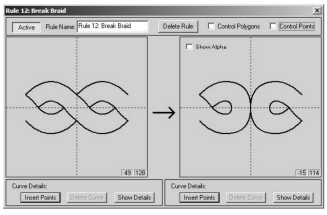

Rule 7: Insert Long Arch L

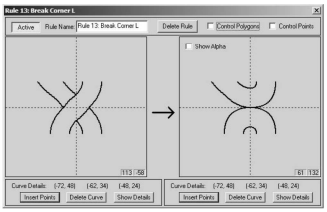

Rule 8: Insert Long Arch R


Rule 9: Insert Corners


Rule 10: Insert Arch and Corner L


Rule 11: Insert Arch and Corner R


Rule 12: Break Braid


Rule 13: Break Corner L

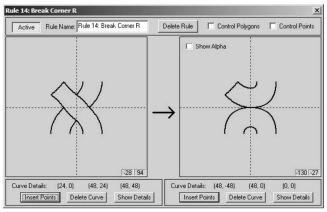

Rule 14: Break Corner R

Figure 20 – Local geometry rules of the Celtic knotwork grammar

The validity of a Celtic knotwork pattern can be defined according to two key characteristics of the style: the over-under alternating pattern of plaitwork, and the closed nature of the patterns. Allen (1904) attributes the geometric perfection of knotwork patterns to these characteristics and notes that "every cord laps under and over with unfailing regularity (never over two or under two) and all the cords are joined up so as not to leave any loose ends" (page 256). The shape rules in the Celtic knotwork grammar ensure that these characteristics are always present in a generated design by recognising the local topology of detected subshapes, and making appropriate shape replacements. For example, Rule 3 and Rule 4, illustrated in figure 19 are variations of a single rule that adds a twist to the corner of a knotwork pattern. In order to ensure the over-under alternating pattern of the plaitwork the rule takes into consideration the local topology of the corner and adds an appropriate twist – Rule 3 adds a twist with a left-handed intersection, while Rule 4 adds a twist with a right-handed intersection. This distinction between left- and right-handed intersections is further exemplified in Rules 7 and 8, Rules 10 and 11, and Rules 13 and 14, in figure 20. Similarly, the closure of generated patterns is

ensured because all of the rules maintain the local closure of a design. As a result, any design that is generated from a closed initial shape will also be closed.

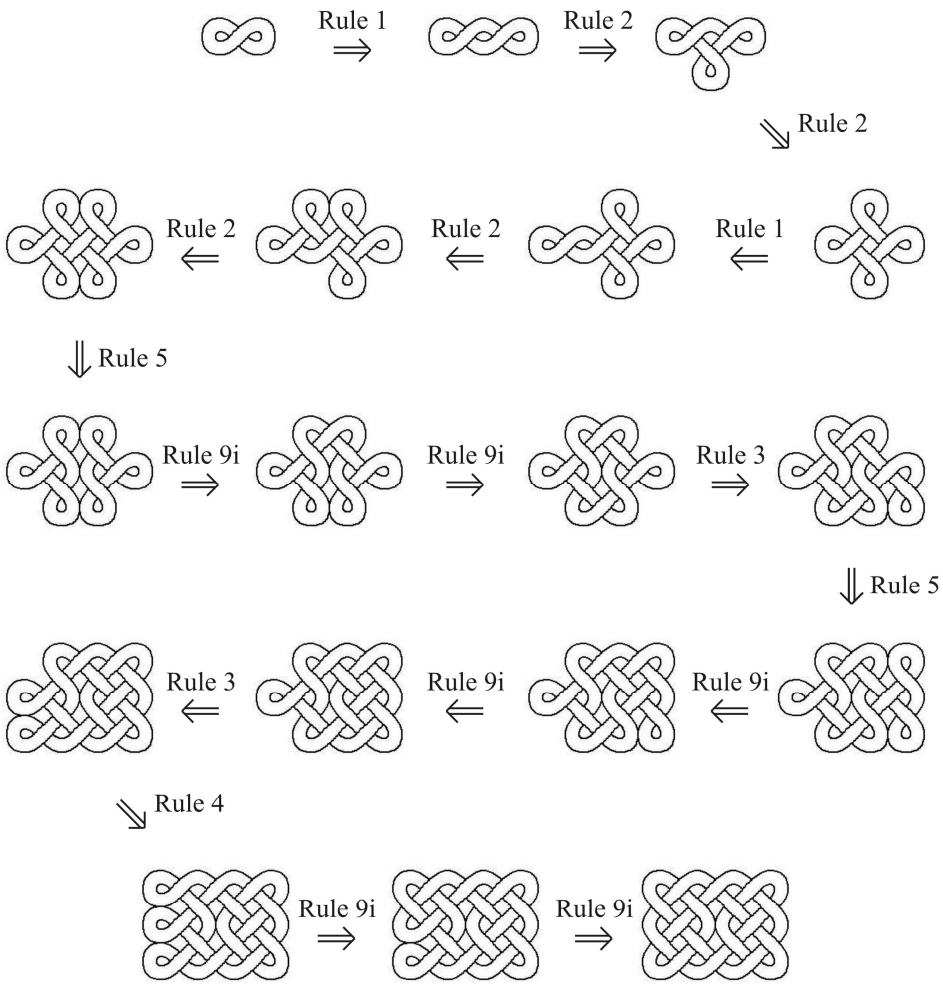


Figure 21 – Generating a Celtic knotwork pattern

The simple procedure of "growing" knotwork patterns, as illustrated in figure 21, offers an elegant new way to conceptualise and generate knotwork designs. It can produce a range of complex designs, including historical examples of the style. For example, Allen (1904) defined eight elementary knots which form the basis of nearly all of the knotwork patterns used in Celtic arts, and Bain (1990) defined an additional two knots. Each of these can be generated using the Celtic knotwork grammar, and as a result, so can the majority of historical examples. Figure 22 illustrates a selection of six designs that were generated using the grammar. Five of these designs are recreations of patterns that were used as ornamentation over a century ago, and the sixth design is an original creation.

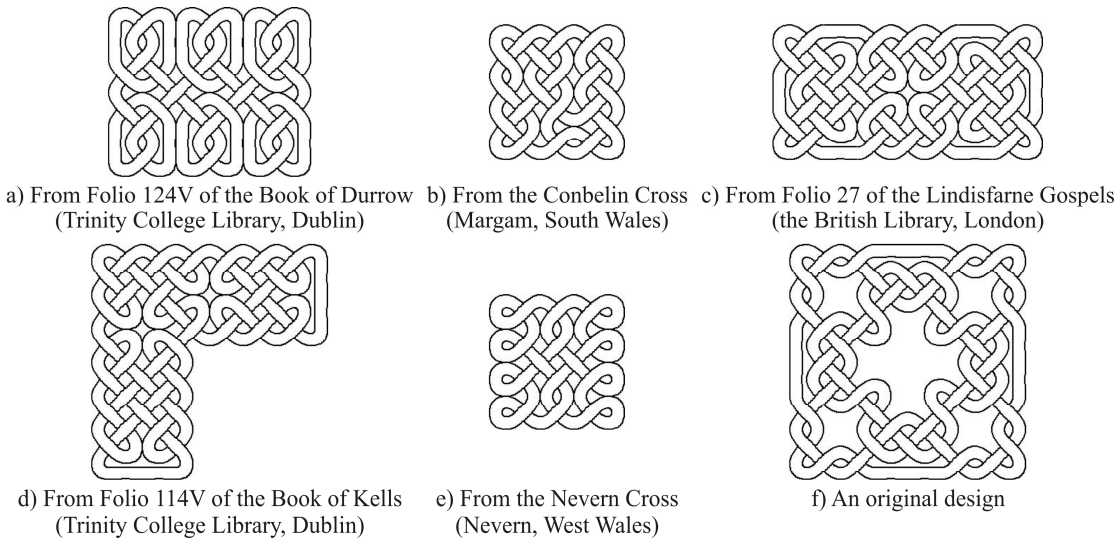


Figure 22 – Examples of knotwork patterns generated using the Celtic knotwork grammar

5. Discussion

In this paper a shape grammar implementation has been presented in which shapes, and the embedding properties of shapes, are not defined according to straight lines but instead are defined according to parametric curve segments. In this respect, the implementation differs from the shape grammar interpreter introduced by McCormack and Cagan (2003), in which the embedding properties of curved shapes are calculated based on those of representative straight line shapes. Curved shape grammar implementations increase the scope of shape grammar application to design disciplines in which curves are predominant. This was illustrated in Section 4, where rules in the Celtic knotwork grammar are applied by recognising embedded curve segments in a manner previously not afforded by shape grammar interpreters. However, as discussed by Jowers and Earl (2010), consideration of curved shapes gives rise to issues previously unexplored with respect to the shape grammar formalism.

Traditionally, the language of shape grammars has been a language of straight lines and rectilinear form (Stiny, 1980), and this has led to an incomplete formalism. In particular, extending the language of shape grammars to incorporate curved shapes highlights difficulties with respect to the embedding properties of shapes. Embedding is a key concept in shape grammars; for example, the subshape relation and shape identity are both defined according to embedding, as follows. A shape α is defined to be a subshape of a shape γ, denoted $\alpha \leq \gamma$ where $\leq$ is the subshape relation, if all of the geometric elements that compose α are found to be embedded in γ. Similarly, a shape α is defined to be identical to a shape γ if both $\alpha \leq \gamma$ and $\gamma \leq \alpha$.

The concept of embedding is linked intuitively to notions of shape similarity. As a result, for shapes composed of straight lines, or other rectilinear geometric elements, embedding can simply be defined according to Euclidean transformations. This is because, if two rectilinear shapes can be mapped onto each other under a Euclidean transformation then they are visual similar, and conversely if two such shapes are visually similar then they can usually be mapped onto each other under a Euclidean transformation. For curved shapes this correspondence between visual similarity and Euclidean transformations does not necessarily hold. For example, the Celtic knotwork grammar makes use of geometric elements that are visually similar to straight lines, e.g. in Rules 6 – 8 (figure 20), but are in fact defined as parabolic curves with a very low curvature. Despite the visual similarity these parabolic curves cannot be mapped onto straight lines under Euclidean transformations. Similarly, the two curve segments C_1 and C_2 illustrated in figure 23 are visually similar, but they cannot be mapped onto each other under a Euclidean transformation because their descriptors are mathematically distinct, as illustrated by the extended curves.

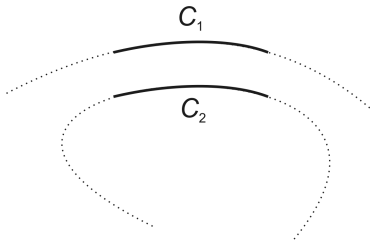


Figure 23 – Visually similar curve segments

The differences in embedding properties between rectilinear shapes and curved shapes arise for two distinct reasons. Firstly, when defined under Euclidean transformations, the embedding relation between curves is limited, as discussed in Jowers and Earl (2009). Under a Euclidean definition, the embedding relation between straight lines is infinite since a line can be embedded in any other line in an infinite number of ways. Conversely, under the same definition, the embedding relation between curve segments is typically finite, and this means that a curve can be embedded in another curve in only a limited number of ways. For example, a quadratic Bézier curve can be embedded in any other quadratic Bézier curve in at most two distinct ways under a Euclidean transformation. This is because quadratic Bézier curves have parabolic descriptors which exhibit a reflective symmetry. Other curve segments with descriptors that do not exhibit such symmetric properties are likely to have embedding properties that are limited further. However, when defined according to transformations other than Euclidean, it is possible for the embedding properties between rectilinear shapes and curved shapes to

be equivalent, as discussed in Jowers (2006). For example, when defined under affine transformations (Euclidean transformations augmented by shear and non-isotropic scale), the embedding properties of quadratic Bézier curves are comparable to the embedding properties of straight lines. This is because, under affine transformations, a parabolic curve can be embedded in any other in an infinite number of ways. Despite this, alternative definitions of embedding are rarely employed in shape grammar applications, and a possible reason for this is that the extent to which alternative definitions of embedding reflect intuitive notions of shape similarity remain to be explored.

Secondly, not all curve segments share the same embedding properties. This is different to the case for straight lines, where all straight lines have the same embedding properties and as a result can always be embedded in each other. The embedding properties of geometric elements are dependent on the descriptors of the elements, and since the descriptors of straight lines are themselves straight lines this results in the infinite embedding relation discussed above. Conversely, the descriptors of curve segments vary, depending on their formal definition. For example, the descriptor of a quadratic Bézier curve is a parabola, while the descriptor of a circular arc is a circle. Under Euclidean transformations these descriptors cannot be mapped onto each other and as a result a quadratic Bézier curve cannot be embedded in a circle, and this means that their embedding properties are distinct. Intuitively, this seems sensible since it results in a differentiation between different types of geometric elements and potential difficulties that could arise, for example as a result of topological differences between descriptors, are avoided. However, issues remain with respect to the link between embedding and intuitive notions of shape similarity. For example, this was illustrated in Figure 23, where C_1 and C_2 are visually similar, but their embedding properties are distinct because their descriptors are mathematically distinct.

This discussion suggests a need to reconsider embedding and other derivative aspects of the shape grammar formalism with curved shapes in mind. In particular, key issues that arise when considering the embedding relation applied to curved shapes include:

- the distinction between visual similarity and formal definitions of similarity
- the different embedding properties that arise when the embedding relation is considered under transformations other than the Euclidean transformations
- the range of embedding properties that arise as a result of fundamental differences between different kinds of curves, e.g. the topological differences between open and closed curves

Stiny (2006) defined algebras of shape, ordered by the subshape relation, and these formalise the shapes, shape operations and spatial transformations used in shape grammar applications. However, their definition is based solely on consideration of rectilinear shapes. These algebras need to be readdressed in light of the issues that arise as a result of exploring the properties of curved shapes. This in turn would result in a better understanding of how curved shapes can be incorporated into the shape grammar formalism.

References

Allen J R, 1904, *Celtic Art in Pagan and Christian Times* (Methuen & Co, London)

Bain G, 1975, *Celtic Art: The Method of Construction* (Lewis Reprints Ltd, London)

Bain I, 1990, *Celtic Knotwork* (BAS Printers Ltd, Hampshire)

Brown K N and Cagan J, 1997, "Optimized process planning by generative simulated annealing" *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 11(3) 219–235

Chase S C, 1989, "Shape and shape grammars - from mathematical model to computer implementation" *Environment and Planning B-Planning and Design* **16**(2) 215–242

Chase S C, 2002, "A model for user interaction in grammar-based design systems" *Automation in Construction* 11(2) 161-172

Do Carmo M P, 1976 Differential Geometry of Curves and Surfaces (Prentice-Hall, Englewood Cliffs, NJ)

Farin G, Hoschek J and Kim M S, 2002 *Handbook of Computer Aided Geometric Design* (Elsevier, Amsterdam)

Jablan S V, 2002 *Symmetry, Ornament, and Modularity* (World Scientific Publishing. London)

Jowers I, 2006 *Computation with Curved Shapes: Towards Freeform Shape Generation in Design*. PhD Thesis, The Open University

Jowers I, Earl C, 2010, "The construction of curved shapes" Environment *and Planning B-Planning and Design* 37(1) 42 – 58

Knight T, Sass L, In Press, "Looks Count: Computing and Constructing Visually Expressive Mass Customized Housing", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*

Koning H and Eizenberg J, 1981 "The language of the prairie - Frank Lloyd Wright's Prairie houses" *Environment and Planning B-Planning and Design* **8**(3) 295–323

Krishnamurti R, 1981, "The construction of shapes" *Environment and Planning B-Planning and Design* **8**(1) 5–40

McCormack J P and Cagan J, 2003, “Increasing the scope of implemented shape grammars: a shape grammar interpreter for curved shapes” in *ASME 2003 International DETC and Computers and Information in Engineering Conference*, Chicago, Illinois

Orsborn S, Cagan J, Pawlicki R, Smith R, 2006, “Creating Cross-over Vehicles: Defining and Combining Vehicle Classes Using Shape Grammars,” *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 20(3) 217-246.

Stiny G, 2006 *Shape: Talking about Seeing and Doing* (MIT Press, Cambridge, Massachusetts)

Tapia M, 1999, “A visual implementation of a shape grammar system” *Environment and Planning B-Planning and Design* 26(1) 59–74

Trescak T, Rodriguez I, Esteva M, 2009, ‘General Shape Grammar Interpreter for Intelligent Designs Generations’ in *Proceedings of Sixth International Conference on Computer Graphics, Imaging and Visualization*, Tianjin, China, pp 235-240